2022

AP® Computer Science A

Scoring Guidelines

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

Question 1: Methods and Control Structures — 9 points

Canonical solution

(a) — **4 points**

```
public int getScore()
{
   int score = 0;

   if (levelOne.goalReached())
   {
      score = levelOne.getPoints();

      if (levelTwo.goalReached())
      {
         score += levelTwo.getPoints();

         if (levelThree.goalReached())
         {
            score += levelThree.getPoints();
         }
      }
   }

   if (isBonus())
   {
      score *= 3;
   }

   return score;
}
```

(b) — **5 points**

```
public int playManyTimes(int num)
{
   int max = 0;

   for (int i = 0; i < num; i++)
   {
      play();
      int score = getScore();
      if (score > max)
      {
         max = score;
      }
   }

   return max;
}
```

(a) `getScore`

	Scoring Criteria	Decision Rules	
1	Calls `getPoints`, `goalReached`, and `isBonus`	Responses **will not** earn the point if they • fail to call `getPoints` or `goalReached` on a `Level` object • call `isBonus` on an object other than `this` (use of `this` is optional) • include parameters	**1 point**
2	Determines if points are earned based on `goalReached` return values	Responses **can** still earn the point even if they • calculate the score total incorrectly • call `goalReached` incorrectly • fail to distinguish all cases correctly Responses **will not** earn the point if they • fail to use a nested `if` statement or equivalent	**1 point**
3	Guards update of score for bonus game based on `isBonus` return value	Responses **can** still earn the point even if they • triple the calculated score incorrectly • update the score with something other than tripling • call `isBonus` incorrectly Responses **will not** earn the point if they • use the `isBonus` return value incorrectly	**1 point**
4	Initializes and accumulates appropriate score (*algorithm*)	Responses **can** still earn the point even if they • call methods incorrectly, as long as method calls are attempted • fail to return the score (*return is not assessed*) Responses **will not** earn the point if they • calculate the score total incorrectly • triple the calculated score incorrectly	**1 point**
		Total for part (a)	**4 points**

(b) `playManyTimes`

	Scoring Criteria	Decision Rules	
5	Loops `num` times	Responses **can** still earn the point even if they • return early	**1 point**
6	Calls `play` and `getScore`	Responses **will not** earn the point if they • call either method on an object other than `this` (use of `this` is optional) • include parameters	**1 point**
7	Compares a score to an identified max or to another score	Responses **can** still earn the point even if they • make the comparison outside the loop • call `getScore` incorrectly • fail to call `play` between calls to `getScore`	**1 point**
8	Identifies the maximum score (*algorithm*)	Responses **will not** earn the point if they • fail to initialize the result variable • compare a score to an identified max or to another score outside the loop • fail to call `play` exactly once each time through the loop	**1 point**
9	Returns identified maximum score	Responses **can** still earn the point even if they • calculate the maximum score incorrectly Responses **will not** earn the point if they • assign a value to the identified maximum score without any loop or logic to find the maximum	**1 point**
		Total for part (b)	**5 points**

Question-specific penalties
None

Total for question 1 **9 points**

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

Question 2: Class — 9 points

Canonical solution

9 points

```
public class Textbook extends Book
{
   private int edition;

   public Textbook(String tbTitle, double tbPrice,
                   int tbEdition)
   {
      super(tbTitle, tbPrice);
      edition = tbEdition;
   }

   public int getEdition()
   {
      return edition;
   }

   public boolean canSubstituteFor(Textbook other)
   {
      return other.getTitle().equals(getTitle()) &&
             edition >= other.getEdition();
   }

   public String getBookInfo()
   {
      return super.getBookInfo() + "-" + edition;
   }
}
```

`Textbook`

	Scoring Criteria	Decision Rules	
1	Declares class header (must not be `private`): `class Textbook extends Book`		**1 point**
2	Declares constructor header: `public Textbook(String ___, double ___, int ___)`		**1 point**
3	Constructor calls `super` as the first line with the appropriate parameters		**1 point**
4	Declares appropriate `private` instance variable and uses appropriate parameter to initialize it	Responses **will not** earn the point if they • omit the keyword `private` • declare the variable outside the class, or in the class within a method or constructor • redeclare and use the instance variables of the superclass	**1 point**

5	Declares at least one required method and all declared headers are correct:
	`public boolean canSubstituteFor(Textbook ___)`
	`public int getEdition()`
	`public String getBookInfo()`
		Responses **will not** earn the point if they
		• exclude `public`
			1 point

6	`getEdition` returns value of instance variable	Responses **will not** earn the point if they • fail to create an instance variable for the edition	**1 point**
7	`canSubstituteFor` determines whether `true` or `false` should be returned based on comparison of book titles and editions (*algorithm*)	Responses **can** still earn the point even if they • fail to return (*return is not assessed for this method*) • access the edition without calling `getEdition` • redeclare and use the `title` variable of the superclass instead of calling `getTitle` Responses **will not** earn the point if they • fail to use `equals` • call `getTitle` incorrectly in either case	**1 point**
8	`getBookInfo` calls `super.getBookInfo`	Responses **can** still earn the point even if they • redeclare and use the instance variables of the superclass Responses **will not** earn the point if they • include parameters	**1 point**

9	Constructs information string	Responses **can** still earn the point even if they • call `super.getBookInfo` incorrectly • fail to call `super.getBookInfo` and access `title` and `price` directly • fail to return (*return is not assessed for this method*) Responses **will not** earn the point if they • omit the literal hyphen(s) in the constructed string • omit the edition in the constructed string • concatenate strings incorrectly	**1 point**

Question-specific penalties
None

Total for question 2 9 points

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

Question 3: Array / ArrayList — 9 points

Canonical solution

(a) — **3 points**

```
public double getAverageRating()
{
   int sum = 0;

   for (Review r : allReviews)
   {
      sum += r.getRating();
   }

   return (double) sum / allReviews.length;
}
```

(b) — **6 points**

```
public ArrayList<String> collectComments()
{
   ArrayList<String> commentList = new ArrayList<String>();

   for (int i = 0; i < allReviews.length; i++)
   {
      String comment = allReviews[i].getComment();
      if (comment.indexOf("!") >= 0)
      {
         String last =
            comment.substring(comment.length() - 1);
         if (!last.equals("!") && !last.equals("."))
         {
            comment += ".";
         }
         commentList.add(i + "-" + comment);
      }
   }
   return commentList;
}
```

(a) `getAverageRating`

	Scoring Criteria	Decision Rules	
1	Initializes and accumulates sum	Response **can** still earn the point even if they • fail to use a loop to accumulate • fail to call `getRating` or call `getRating` incorrectly	**1 point**
2	Accesses every element of `allReviews` (*no bounds errors*)	Responses **will not** earn the point if they • access the elements of `allReviews` incorrectly	**1 point**
3	Computes and returns `double` average rating based on `getRating` return values (*algorithm*)	Response **can** still earn the point even if they • fail to initialize the accumulator for the sum Responses **will not** earn the point if they • fail to accumulate the sum of all ratings • use integer division to compute average • include parameters on call to `getRating` • fail to call `getRating` on all elements of `allReviews`	**1 point**
		Total for part (a)	**3 points**

(b) `collectComments`

	Scoring Criteria	Decision Rules	
4	Instantiates an `ArrayList` capable of holding `String` objects		**1 point**
5	Accesses every element of `allReviews` (*no bounds errors*)	Responses **can** still earn the point even if they • fail to keep track of the index Responses **will not** earn the point if they • access the elements of `allReviews` incorrectly	**1 point**
6	Calls `getComment` on an element of `allReviews`, calls at least one `String` method appropriately on the `getComment` return value, and all `String` method calls are syntactically	Responses **can** still earn the point even if they • call some of the `String` methods on objects other than `getComment` return values Responses **will not** earn the point if they • include a parameter when calling `getComment` • call any `String` methods incorrectly • call any `String` methods on objects other than `String` values	**1 point**
7	Compares the final character of the comment to both a period and an exclamation point	Responses **can** still earn the point even if they • use incorrect logic in the comparison • call `String` methods incorrectly Responses **will not** earn the point if they • use `==` instead of `equals` when comparing `String` objects	**1 point**
8	Assembles string appropriately based on result of comparison of last character with period and exclamation point (*algorithm*)	Responses **can** still earn the point even if they • call `String` methods incorrectly • use `==` instead of `equals` Responses **will not** earn the point if they • fail to keep track of the element index • use incorrect logic in the comparison	**1 point**
9	Adds all and only appropriate constructed strings to the `ArrayList` (*algorithm*)	Responses **can** still earn the point even if they • initialize the `ArrayList` incorrectly • fail to return the constructed `ArrayList` (*return is not assessed*) • assemble the review string incorrectly • access the elements of `allReviews` incorrectly	**1 point**

Total for part (b)	**6 points**

Question-specific penalties
None

Total for question 3	**9 points**

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

Question 4: 2D Array — 9 points

Canonical solution

(a) — **4 points**

```
public void repopulate()
{
   for (int row = 0; row < grid.length; row++)
   {
      for (int col = 0; col < grid[0].length; col++)
      {
         int rval = (int)(Math.random() * MAX) + 1;
         while (rval % 10 != 0 || rval % 100 == 0)
         {
            rval = (int)(Math.random() * MAX) + 1;
         }
         grid[row][col] = rval;
      }
   }
}
```

(b) — **5 points**

```
public int countIncreasingCols()
{
   int count = 0;

   for (int col = 0; col < grid[0].length; col++)
   {
      boolean ordered = true;

      for (int row = 1; row < grid.length; row++)
      {
         if (grid[row][col] < grid[row-1][col])
         {
            ordered = false;
         }
      }

      if (ordered)
      {
         count++;
      }
   }

   return count;
}
```

(a) `repopulate`

	Scoring Criteria	Decision Rules	
1	Traverses `grid` (*no bounds errors*)	Responses **will not** earn the point if they • fail to access an element of `grid` • access the elements of `grid` incorrectly • use enhanced `for` loops without using a `grid` element inside the loop	**1 point**
2	Generates a random integer in a range based on `MAX`	Responses **can** still earn the point even if they • assume or verify that `MAX >= 10` Responses **will not** earn the point if they • fail to cast to an `int`	**1 point**
3	Ensures that all produced values are divisible by 10 but not by 100	Responses **can** still earn the point even if they • fail to use a loop	**1 point**
4	Assigns appropriate values to all elements of `grid` (*algorithm*)	Responses **can** still earn the point even if they • assume or verify that `MAX >= 10` • produce some values that are not divisible by 10 or divisible by 100, if the range and distribution are otherwise correct Responses **will not** earn the point if they • use enhanced `for` loops and fail to maintain indices • produce values that are not equally distributed • produce values outside the specified range • exclude values that should be considered valid (other than errors in 10/100 handling)	**1 point**
		Total for part (a)	**4 points**

(b) `countIncreasingCols`

	Scoring Criteria	Decision Rules	
5	Traverses `grid` in column major order (*no loop header bounds errors*)	Responses **can** still earn the point even if they • access an out-of-bounds row or column index adjacent to the edge of the grid, if the loop bounds include only valid indices Responses **will not** earn the point if they • fail to access an element of `grid` • access the elements of `grid` incorrectly	**1 point**
6	Compares two elements in the same column of `grid`	Responses **can** still earn the point even if they • access elements of `grid` incorrectly • access elements in nonadjacent rows • compare elements with `==` • compare two elements in the same row instead of the same column	**1 point**
7	Determines whether a single column is in increasing order (*algorithm*)	Responses **can** still earn the point even if they • fail to reset variables in the outer loop before proceeding to the next column Responses **will not** earn the point if they • fail to access all pairs of adjacent elements in a single column • cause a bounds error by attempting to compare the first element of a column with a previous element or the last element of a column with a subsequent element • incorrectly identify a column with at least one pair of adjacent elements in decreasing order as increasing	**1 point**
8	Counts all columns that are identified as increasing (*algorithm*)	Responses **can** still earn the point even if they • detect increasing order for each row instead of each column • incorrectly identify increasing columns in the inner loop Responses **will not** earn the point if they • fail to initialize the counter	**1 point**

		• fail to reset variables in the outer loop causing subsequent runs of the inner loop to misidentify columns	
9	Returns calculated count of increasing columns	Responses **can** still earn the point even if they • calculate the count incorrectly	**1 point**
		Total for part (b)	**5 points**
	Question-specific penalties		
	None		
		Total for question 4	**9 points**

Alternate Canonical for Part (a)

```
public void repopulate()
{
   for (int row = 0; row < grid.length; row++)
   {
      for (int col = 0; col < grid[0].length; col++)
      {
         int rval = ((int)(Math.random() * (MAX / 10)) + 1) * 10;
         while (rval % 100 == 0)
         {
            rval = ((int)(Math.random() * (MAX / 10)) + 1) * 10;
         }
         grid[row][col] = rval;
      }
   }
}
```